VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVÁNÍ RUČNĚ PSANÉHO PÍSMA POMOCÍ NEURONOVÝCH SÍTÍ

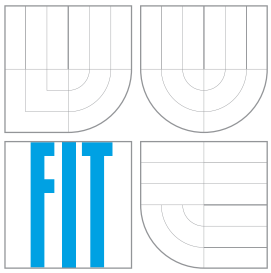
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH SMEJKAL

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVÁNÍ RUČNĚ PSANÉHO PÍSMA POMOCÍ NEURONOVÝCH SÍTÍ

HANDWRITTEN CHARACTER RECOGNITION USING ARTIFICIAL NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VOJTĚCH SMEJKAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. OLDŘICH PLCHOT
SUPERVISOR

BRNO 2011

Abstrakt

Práce se zabývá rozpoznáváním velkých písmen a číslic hůlkového písma pomocí neuronových sítí. Rozebírá použité algoritmy pro segmentaci textu, metody extrakce příznaků a problematiku učení sítě pomocí zpětného šíření chyb. Jsou zde také popsány provedené experimenty s různými konfiguracemi nad datovými sadami. Pro trénování nových sítí a testování jejich úspěšnosti byla vytvořena demonstrační aplikace s grafickým rozhraním s možností interaktivního rozpoznávání myší psaného textu.

Abstract

Thesis deals with handwritten block letters and digits recognition using artificial neural networks. Text segmentation algorithms, feature extraction methods and backpropagation learning are explained. There are also described performed experiments with variety of configurations on datasets. Application with graphical user interface and interactive mouse-written text recognition was created to train new neural networks and test their effectivity.

Klíčová slova

umělé neuronové sítě, rozpoznávání vzorů, zpracování obrazu, OCR, extrakce příznaků, segmentace textu, učení zpětným šířením, FANN

Keywords

artificial neural networks, pattern recognition, image processing, OCR, features extraction, text segmentation, backpropagation learning, FANN

Citace

Vojtěch Smejkal: Rozpoznávání ručně psaného písma pomocí neuronových sítí, bakalářská práce, Brno, FIT VUT v Brně, 2011

Rozpoznávání ručně psaného písma pomocí neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Oldřicha Plchota

.....
Vojtěch Smejkal
15. května 2011

Poděkování

Rád bych poděkoval Ing. Oldřichu Plchotovi za cenné rady a odbornou pomoc v průběhu celého projektu, Bc. Jozefu Hrickovi za poskytnuté vyplněné skenovací formuláře a také své rodině za psychickou podporu a motivaci.

Obsah

Kapitola 1

Úvod

OCR systémy a jejich praktické využití zažívají v posledních letech zvýšený zájem veřejnosti a především firem. Jednak díky projektům typu Project Gutenberg, který se zasloužil o převedení více jak 30 000 knih [20] do elektronické podoby a jejich zpřístupnění zdarma celému světu. Podstatným důvodem pro převod obrázků na text je možnost indexování a vyhledávání. To je klíčová schopnost zejména v dnešní době se stále se zvětšujícím objemem informací na internetu.

Tato práce se zabývá klasifikací znaků psaného hůlkového písma pomocí neuronových sítí. Jde tedy o analýzu jednotlivých pixelů znaku a rozhodnutí, o jaké písmeno nebo číslici se s největší pravděpodobností jedná. Tato metoda je součástí problematiky *rozpoznávání vzorů* patřící do oboru tzv. strojového učení, což je jedna z větví umělé inteligence. Klasifikace se snaží zařadit každou vstupní hodnotu do jedné z daných množin tříd. Používá se například pro odlišení nevyžádané pošty od běžné zprávy, rozpoznání jazyka, ve kterém je napsaný text, apod.

V rámci řešení byla vytvořena aplikace s uživatelským rozhraním. Umožňuje interaktivě psát znaky a zobrazovat statistiky každého rozpoznání, trénovat nové neuronové sítě s množstvím nastavitelných parametrů a posléze je testovat. Díky použití interpretovaného jazyka je možné také navrhovat nové algoritmy pro extrakci příznaků přímo ve vestavěném editoru a současně ověřovat jejich reálnou úspěšnost. Cílem tedy bylo vyvinout program, který bude sloužit jako demonstrační i výuková pomůcka.

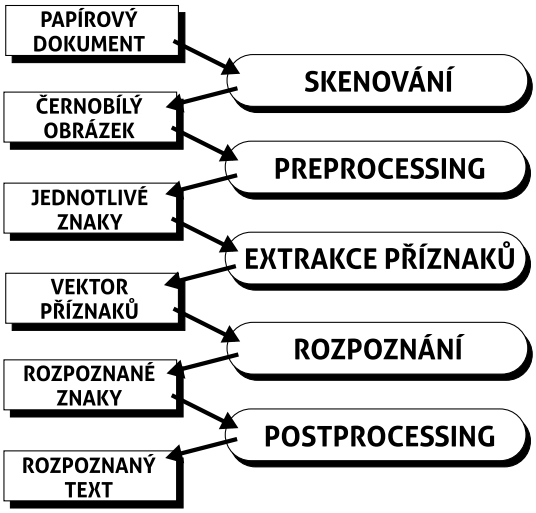
Kapitola 2

Teoretická část

V této kapitole budou formálně vysvětleny základní pojmy z oblasti rozpoznávání vzorů a neuronových sítí.

2.1 OCR

Optické rozpoznávání znaků je komplexní proces skládající se z několika podúloh. Většinou nelze žádnou z těchto úloh vynechat ani přehodit jejich pořadí. Pro praktické využití je třeba také dosáhnout co nejkratšího času rozpoznání. To vyžaduje nejenom každou část co nejvíce optimalizovat na výkon, ale také použít vhodné algoritmy. Schéma 2.1 znázorňuje typický průběh OCR:



Obrázek 2.1: Jednotlivé kroky v procesu OCR [5]

1. **Skenování**
   Provádí převod fyzického textu na černobílý obrázek. Je důležité dodržet určité rozlišení, aby nedošlo k přílišné pixelizaci obrazu.

2. **Preprocessing**
   Předzpracování se provádí před samotným rozpoznáním. Skládá se často z vícero kroků.

(a) Převod černobílého obrázku na binární. Tedy mapování hodnot 0–255 na hodnoty 0–1. Klíčové je zvolit správný práh (treshold).

(b) Segmentace textu na řádky a jednotlivé znaky.

(c) Normalizace znaků na stejnou velikost, popř. pootočení.

3. **Extrakce příznaků**
   Nejdůležitější část procesu, která rozhoduje o celkové úspěšnosti rozpoznání. Jedná se o extrakci relevantní informace z binárních dat. Vhodné metody minimalizují rozdíly mezi vzorky jedné třídy, zatímco zvyšují rozdílnost mezi vzorky jiných tříd. [4]

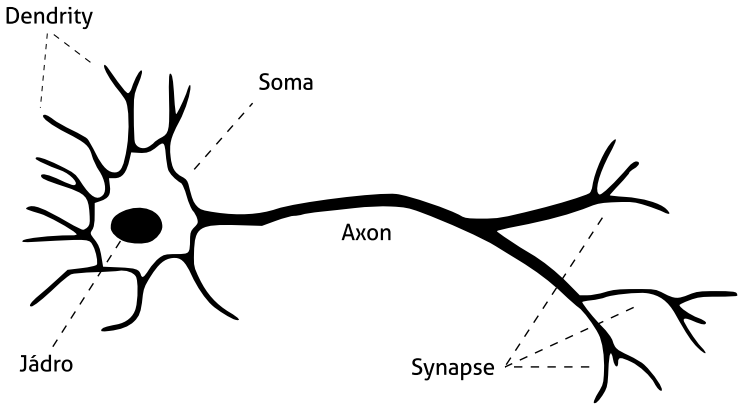
4. **Rozpoznání**
   Příznaky se vloží na vstup klasifikátoru. Ten podle nich zařadí znak do příslušné třídy. Pokud rozpoznáváme písmena anglické abecedy a číslice, je těchto tříd 36.

5. **Postprocessing**
   Získané znaky složíme zpět znovu do slov a vět, kde se uplatní algoritmy pro detekci mezer mezi slovy. Je možné provést kontextovou verifikaci, která ověřuje smysluplnost znaku v daném textu. Zabraňuje například tomu, aby se v rodném čísle objevilo písmeno, apod.

2.2 Neuronové sítě

Informace k této kapitole byly čerpány ze zdrojů [9], [2] a [18].

Jedním z klasifikátorů použitelným pro rozpoznávání vzorů jsou neuronové sítě (NS). Tyto struktury zpočátku vznikaly jako zjednodušené modely skutečných biologických neuronových sítí. Jejich základem je biologický neuron znázorněný na obrázku 2.3, který tvoří tři hlavní části: *dendrity*, *soma* a *axon*.



Obrázek 2.2: Biologický neuron [14]

Pomocí dendritů neuron přijímá signály, v části soma je zpracuje a výsledný impulz pošle axonem dále na výstup. Na axon jsou napojeny synapse, které mohou dále signál zesilovat nebo zeslabovat. Na tomto konceptu jsou založeny i *umělé neuronové sítě* popisované dále.

NS se zásadně liší od ostatních prostředků stejného typu. Nelze přesně analyzovat jejich účinnost a proto nemůžeme vysvětlit, proč dala síť za jistých podmínek takový či jiný výsledek. Jejich funkce je závislá na struktuře sítě, na algoritmech pro úpravu vnitřních parametrů nebo na způsobu, jak s nimi experimentujeme. Procesní jednotky (neurony) jsou funkčně *primitivní*, ale za to jich pracuje velký počet najednou. Celá síť funguje asynchronně a dochází k *paralelnímu zpracování*. Neurony jsou mezi sebou hustě propojeny pomocí synapsí s různými *vahami*, které udávají aktuální konfiguraci sítě. NS mají velkou schopnost *adaptability a odolnosti* vůči nepřesnostem ve vstupní informaci. Této klíčové vlastnosti se využívá zejména v rozpoznávání vzorů, které bývají také často deformované nebo nějakým způsobem neúplné.

Typická NS má vstup, na který přivádíme vektor příznaků, a výstup, ze kterého získáváme ohodnocené třídy. Z matematického hlediska se tak dá na NS nahlížet jako na zobrazení vstupního vektorového prostoru x do výstupního vektorového prostoru y (2.1). Dimenze těchto prostorů mohou být obecně různé.

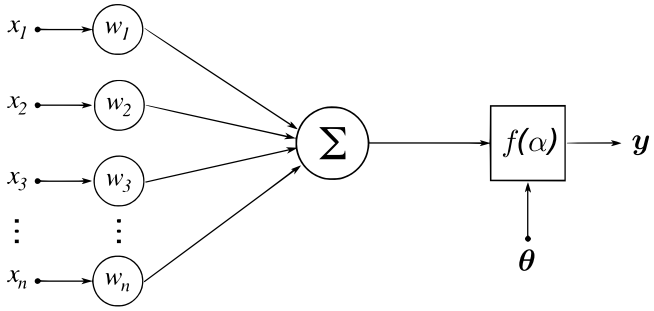
$$\{\mathbf{x}\} \rightarrow \{\mathbf{y}\} \tag{2.1}$$

2.2.1 Umělý neuron

Neuron je procesní prvek, který má N vstupů a jeden výstup. Je charakterizován rovnicí

$$y = f\left(\sum_{i=1}^{N} w_i \; x_i - \theta\right), \tag{2.2}$$

která udává jeho výstup y pro vstupní vektor $\mathbf{x} = [x_1, x_2, \ldots, x_N]^T$, kde $\mathbf{w} = [w_1, w_2, \ldots, w_N]^T$ je vektor aktuálních vah, θ je aktuální práh neuronu a $f(\alpha)$ je neměnná funkce nazývaná charakteristika neuronu nebo také *aktivační funkce*.

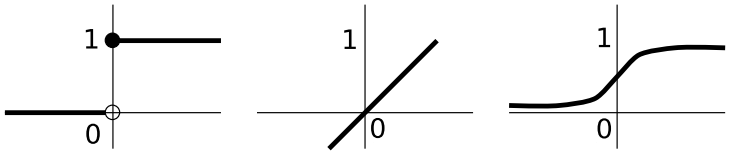


Obrázek 2.3: Schéma umělého neuronu

Běžně používané charakteristiky jsou skoková funkce, lineární funkce a sigmoida, které jsou znázorněny na obrázku 2.4. V dopředných vícevrstvých sítích se využívá výhodných vlastností právě sigmoidní funkce

$$f(\alpha) = \frac{1}{1 + e^{\frac{-\alpha}{T}}}, \tag{2.3}$$

která je spojitá, má spojité derivace a parametr T mění její strmost v okolí nuly. Generuje jemné a přirozené výstupní hodnoty v intervalu $(0, 1)$. Funkce nejvíce reaguje na vstup v intervalu $(-1, 1)$.



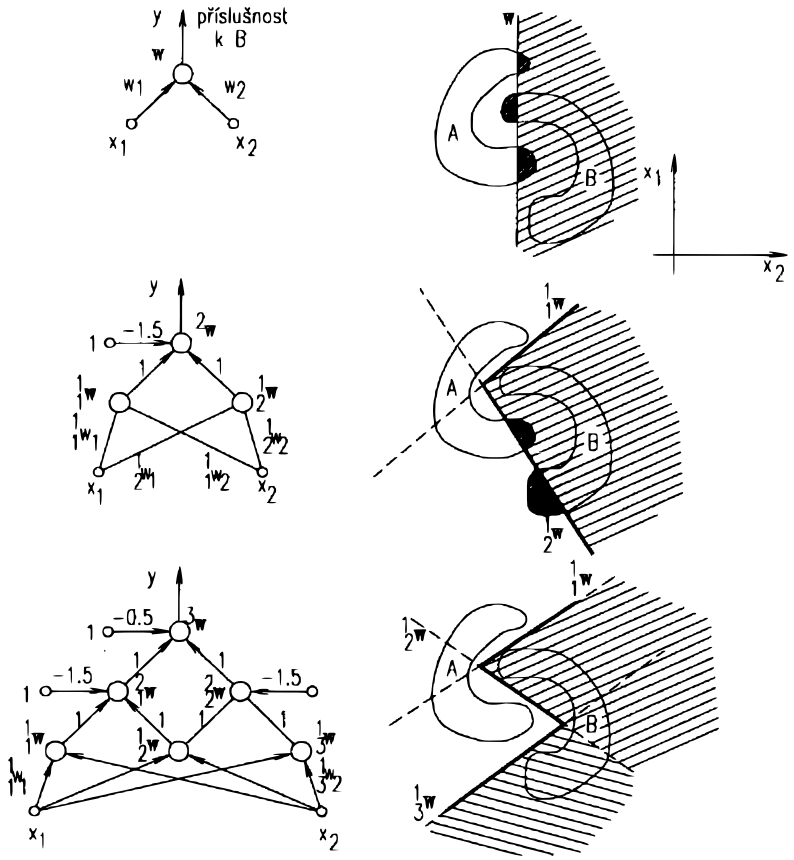
Obrázek 2.4: Aktivační funkce: skoková, lineární a sigmoidní

Existuje řada algoritmů učení neuronů. Typickým pravidlem opravy vah je *δ-pravidlo*

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \mu(y_d - y)\mathbf{x} \ , \tag{2.4}$$

kde y_d je požadovaný výstup neuronu, y je skutečný výstup při vstupu $\mathbf{x}$ a vahách $\mathbf{w}_n$, a μ je *rychlost učení*, konstanta ovlivňující rychlost konvergence.

2.2.2 Architektura sítě



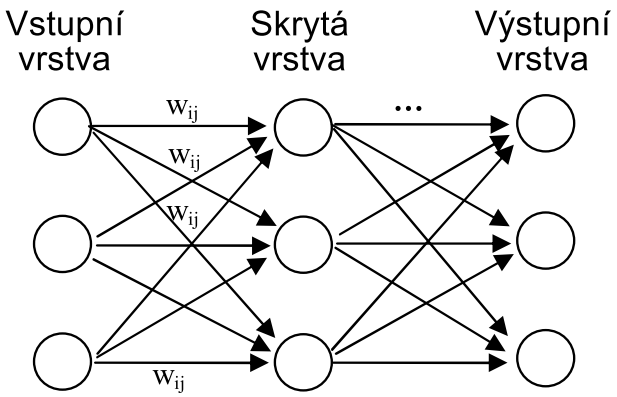
Obrázek 2.5: Klasifikační schopnosti sítě s jednou, dvěma a třemi vrstvami [9]

Nejjednodušší neuronovou sítí je **jednovrstvý perceptron**. Jedná se o řadu M paralelně zapojených neuronů, které realizují transformaci vstupního vektoru, $y_j = f(\mathbf{w}_j^T\mathbf{x})$, podle svého vektoru vah $\mathbf{w}_j$ [9].

Nechť je dán počet složek $N = 2$ vstupního vektoru $\mathbf{x}$. Poté můžeme interpretovat prvky tohoto vektoru $\mathbf{x_1}, \mathbf{x_2}$ jako souřadnice bodu v rovině. Mějme dvě množiny bodů A, B a vstupní vektor, který chceme do jedné z těchto množin zařadit. Jednovrstvý perceptron nedokáže prvky těchto dvou množin za daných podmínek rozlišit, neboť realizuje pouze lineární funkci. I při optimální poloze dělící přímky budou jisté oblasti klasifikovány chybně, tak jak je to znázorněno v horní částí obrázku 2.5.

Pokud přidáme k původní síti další vrstvu neuronů realizující operaci konjunkce, je již výsledek o něco lepší. Dostatečným množstvím polorovin lze ohraničit libovolný konvexní tvar oblasti. Pokud je ale oblast nekonvexní, je nutné přidat ještě jednu vrstvu neuronů, která bude vykonávat operaci disjunkce. Tak můžeme sjednotit všechny průniky dělících nadrovin, jak je vidět ve spodní části obrázku 2.5. Bylo dokázáno, že takováto třívrstvá dopředná neuronová síť dokáže aproximovat jakoukoli obecnou funkci [7].

Výše jmenovaná síť je zvláštním případem **vícevrstvého perceptronu** (obrázek 2.6). Obsahuje *výstupní vrstvu* s M neurony, *skryté vrstvy*, kterých může být obecně více, nejčastěji se však používá pouze jedna, a *vstupní vrstvu* s N uzly. Síť je *úplně propojená*, tzn. výstup každého neuronu je připojen na vstup všech neuronů následující vrstvy. Signály se v ní šíří jedním směrem od vstupní k výstupní vrstvě. Proto se také sítě tohoto typu označují jako *dopředné neuronové sítě*.



Obrázek 2.6: Třívrstvá neuronová síť

2.2.3 Učení zpětným šířením chyb

Stejným způsobem jako jednotlivé neurony se i celá síť učí z trénovací množiny dvojic $(\mathbf{x}_p, \mathbf{d}_p)$, kde $\mathbf{x}_p$ je N-rozměrný vstupní vektor trénovací množiny a $\mathbf{d}_p$ je jemu odpovídající M-rozměrný výstupní vektor. Krok učení probíhá tak, že přivedeme na vstup sítě vektor $\mathbf{x}_p$ a zjistíme, jak se liší získaný výstup sítě

$$\mathbf{y}_p = \overline{\Psi}\left(\mathbf{x}_p\right) \tag{2.5}$$

od požadovaného výstupu $\mathbf{d}_p$. Podle zjištěných odchylek upravíme váhy jednotlivých neuronů buď po několika krocích nebo za celou epochu. *Epocha učení* znamená postupné provádění učebních kroků s užitím všech dvojic z trénovací množiny. Obvykle je třeba vykonat velké množství těchto epoch k zajištění dostatečně malé chybovosti klasifikace.

Protože optimálním způsobem učení neuronu je δ-pravidlo (2.4), chtěli bychom ho využít

k učení celé sítě. Má-li síť n_0 vrstev a vstupní vektor $\mathbf{x}_p$, je chyba j-tého výstupu sítě

$$ {}_j^{n_0}e_p = {}_jd_p - {}_jy_p \ . \tag{2.6} $$

Takto však nelze vypočítat chyby neuronů ve vstupní a ve skrytých vrstvách. Použijeme proto představy, že chyba ${}_j^{n_0}e_p$ kteréhokoli j-tého neuronu ležícího ve vrstvě n se rozdělí podle vah propojení na všech napojené neurony předchozí vrstvy. Poté pro chybu i-tého neuronu předchozí vrstvy platí

$$ {}_i^{n-1}e_p \sim \sum_{j=1}^{N_n} {}_j^{n}e_p \ {}_j^{n}w_i \ , \tag{2.7} $$

kde N_n je počet neuronů v n-té vrstvě a ${}_j^{n}w_i$ je i-tá složka vektoru vah j-tého neuronu v n-té vrstvě.

Každý **krok učení** tedy zahrnuje:

- vystavení vektoru $\mathbf{x}_p$ na vstup sítě a zjištění odezvy $\mathbf{y}_p$
- výpočet chybového vektoru $\mathbf{e}_p = \mathbf{d}_p - \mathbf{y}_p$
- získání chyb všech neuronů pomocí zpětného šíření chyb
- oprava vektoru vah každého neuronu podle δ-pravidla

Kapitola 3

Návrh

V této kapitole bude popsána technika parsování znaků ze skenovacích formulářů a jejich ukládání do datových souborů. Budou vysvětleny algoritmy použité pro extrakci obrazových příznaků, segmentaci textu a skeletonizaci znaků. Také se zaměříme na problematiku trénování neuronových sítí z praktického hlediska.

3.1 Zpracování formulářů

Hlavní zdroj trénovacích dat byl získán ze skenovacích formulářů vyplněných dobrovolníky. Vzor takového formuláře je možné najít v příloze práce [8]. V této sekci bude popsán postup, pomocí něhož byla data z formulářů extrahována.

3.1.1 Normalizace

Formuláře byly naskenované na obyčejném barevném stolním skeneru v dostatečně vysokém rozlišení. Vzniklou sadu obrázků, které mohou být pootočené, různě barevné či jinak rozdílné, je nutné před samotným zpracováním unifikovat.

Grayscaling

Převede obrázek z 24bitového barevného prostoru do 8bitové šedé škály. Tím se eliminují různé barvy propisek a rapidně klesne velikost obrázku.

Detekce rohových čtverců

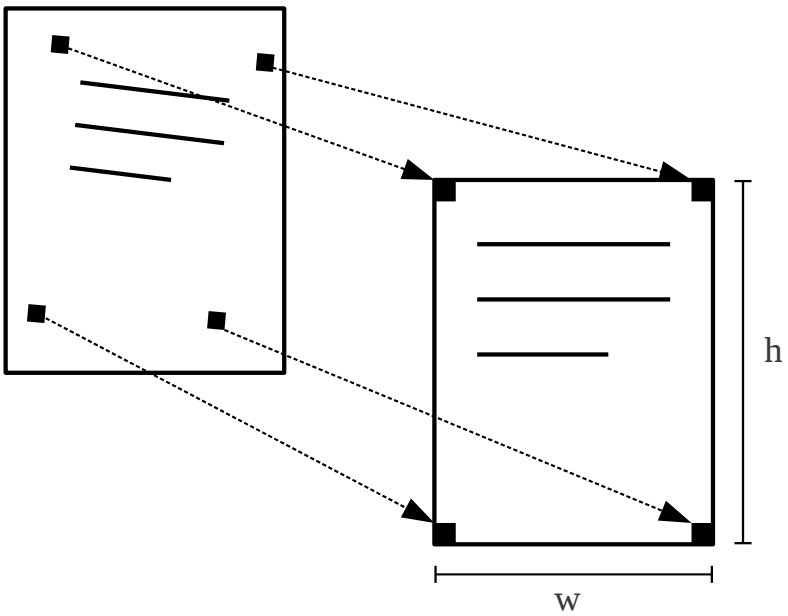
Každý formulář obsahuje ve svých rozích plné černé čtverce, které jsou důležité pro přesné určení polohy obsažených dat. Postup jejich nalezení je popsán v algoritmu 1. Z důvodu optimalizace rychlosti se kontroluje při průchodu formuláře pouze každý n-tý pixel (v tomto případě $n = 5$).

Transformace na obdelník

Pixely uvnitř čtyřúhelníkové oblasti, jejíž vrcholy jsou rohové čtverce, transformujeme na obdelník. Tím otočíme formulář do základní polohy a zmenšíme ho do jednotné velikosti $w \times h$. Operace je znázorněna na obrázku 3.1.

Algoritmus 1 Nalezení rohových čtverců formuláře

```
pro všechny rohy formuláře:
   projít pixely do 1/4 velikosti formuláře:
      pokud je hodnota pixelu < zvolený práh T:
         vypočítat průměr hodnot pixelů v okolí tohoto bodu
         pokud je průměr < práh T:
            čtverec nalezen
            pokračovat na další roh
```



Obrázek 3.1: Mapování oblasti mezi rohovými čtverci na obdelník

3.1.2 Identifikace verze

K rozpoznání typu formuláře slouží identifikační mřížka v pravé horní části formuláře. Jednotlivé buňky mřížky představují bity binárního čísla, které označuje jeho verzi. Podle obsahu buňky určíme, zda představuje binární jedničku (černá výplň) nebo nulu (bílá výplň).

3.1.3 Získání obsahu

Přesné polohy vyplňovacích polí jsou uloženy v programu. Teprve když je formulář normalizovaný, leží tato pole skutečně tam, kde je očekáváme. Známe tedy souřadnice středů těchto polí a použijeme algoritmus 2 k získání jejich obsahu. Konstanty v algoritmu jsou zvoleny takto: $T = 128, N = 30$. Ihned co získáme jedno vyplněné pole z formuláře, spárujeme ho s jeho popiskem. Databázi konkrétních popisků program načte podle identifikačního čísla získaného v předchozím kroku.

Může se stát, že některá pole vyplněná tučným písmem zmatou rozpoznávač. Například tučně napsané písmeno L by mohl algoritmus považovat za levou hranu pole. Pokud se tak stane, získaný výřez nebude čtverový a program ho přeskočí.

Algoritmus 2 Získání obsahu vyplňovacího pole

```
pro směry nahoru, vpravo, dolů, vlevo:
   nastavit pozici na střed políčka
   hrana_nalezena = FALSE
   opakovat dokud není hrana_nalezena:
      procházet pixely v daném směru dokud je jejich hodnota > práh T
      vypočítat průměr hodnot N pixelů kolmých na směr průchodu na obě strany
      pokud průměr hodnot kolmice < práh T:
         uložit pozici hrany
         hrana_nalezena = TRUE
ořízнout obrázek podle uložených pozic
vrátit získaný výřez
```

3.1.4 Způsob uložení

Parsování formulářů je časově náročná operace. Je tedy výhodné si získané dvojice *obrázek–popisek* uložit do datového souboru (datasetu). Inspirací pro formát datasetu byla databáze MNIST [10]. V binárním souboru v kódování **little endian** jsou uloženy:

- počet vzorků `[4 byte]`
- šířka obrázku `[4 byte]`
- výška obrázku `[4 byte]`
- popisek 1 `[1 byte]`
- obrázek 1 `[...]`
- popisek 2 `[1 byte]`
- ...

3.2 Preprocessing vzorků

V této sekci budou vysvětleny postupy předzpracování obrázků znaků předtím, než se na ně aplikují metody extrakce příznaků popsané v sekci 3.3.

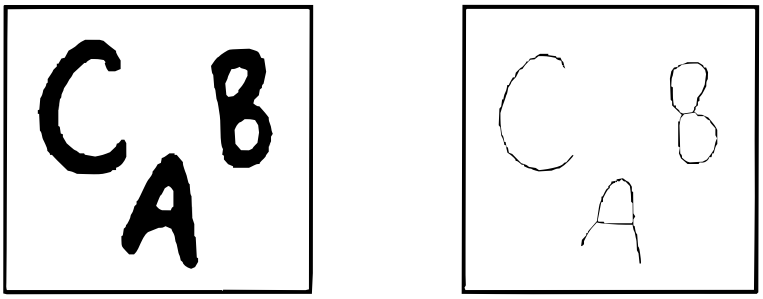
3.2.1 Binarizace

Všechny zde uvedené metody pro získání obrazových příznaků vyžadují na svém vstupu *binární obrázek*. Pixely takového obrázku obsahují pouze dvě barvy: černou a bílou. Barevná paleta tedy neobsahuje stupně šedi $\langle 0, 255 \rangle$, ale pouze hodnoty $\{0, 1\}$, kde nula reprezentuje bílou barvu (pozadí) a jednička černou barvu (popředí).

Obrázek ve stupních šedi je převeden na binární pomocí **prahovací funkce** 3.1 s parametrem T. V tomto případě bylo zvoleno $T = 200$, protože při této hodnotě vzniklý tvar nejlépe vystihuje původní vzhled znaku.

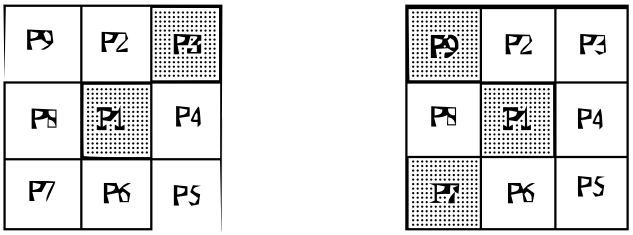
$$f(x) = \begin{cases} 0 & : \quad x \geq T \\ 1 & : \quad x < T \end{cases} \tag{3.1}$$

3.2.2 Skeletonizace



Obrázek 3.2: Skeletonizace: vlevo původní obrázek, vpravo ztenčený

Jeden z faktorů podílejících se na rozdílnosti vzorků patřících do jedné třídy je tloušťka pera. Ke ztenčení čar na velikost jednoho pixelu se využívá metoda zvaná skeletonizace, která je znázorněna na obrázku 3.2. Mezi příbuzné operace patří *eroze* (ztenčování linií) a *dilatace* (rozšiřování linií). V této práci byl pro skeletonizaci použit **Rosenfeldův algoritmus** [3] popsaný v pseudokódu 3.



Obrázek 3.3: P1 jako koncový bod; P1, který není 8-simple

Metoda pracuje s binárními obrázky a prochází vzorek do té doby, dokud je možné nějaký černý pixel odstranit. V tomto algoritmu se pracuje se dvěma pojmy, které je třeba nejprve vysvětlit. *Koncový bod* je takový pixel, který má ve svém osmiokolí pouze jednoho souseda. Tvoří tedy vždy konec nějaké linie. Pojem *8-simple* označuje pixel, který může být odstraněn, aniž by porušil propojení sousedních pixelů. Tyto pojmy jsou ilustrovány na obrázku 3.3

Každý krok ztenčování se skládá ze čtyř průchodů vždy v opačných směrech, např. průchod zleva doprava je následovaný průchodem zprava doleva. V nich se pokaždé odstraní pouze hraniční pixely v daném směru. Tím je zajištěno, že se objekty nesmažou úplně a jejich skeleton bude procházet co nejblíže jejich středu.

Algoritmus 3 Rosenfeldův algoritmus skeletonizace

```
pixel_deleted = TRUE

opakuj dokud je pixel_deleted:
   pixel_deleted = FALSE
   pro směry nahoru, dolů, vpravo, vlevo:
      pro všechny pixely px:
         pd = sousední pixel px v daném směru
         pokud px == 0 nebo pd != 0:
            pokračuj na další pixel
         pokud px je koncový bod:
            pokračuj na další pixel
         pokud není px 8-simple:
            pokračuj na další pixel
         označit px ke smazání
         pixel_deleted = TRUE
      smazat označené pixely
```

3.3 Extrakce příznaků

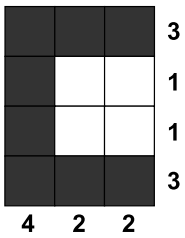
Obrazové příznaky jsou takové informace, které se snaží zdůraznit jedinečné rysy daného znaku. V ideálním případě odlišují jednotlivé třídy písmen a číslic mezi sebou, zatímco minimalizují rozdíly v rámci jedné třídy. Volba vhodných metod pro získání příznaku je základem úspěchu každého OCR. Výstupem metody je vektor příznaků, který je poté přiveden na vstup neuronové sítě. Pokud není řečeno jinak, všechny metody pracují s binárními obrázky.

3.3.1 Pixely

Nejjednodušší metoda spočívá v pouhé serializaci pixelů matice obrázku do vektoru příznaků. Pokud předtím ještě provedeme skeletonizaci a mírné rozostření, můžeme dosáhnout i s takto prostou metodou vysoké úspěšnosti. Negativem je velikost vytvořeného vektoru. Obrázek velikosti $n \times n$ pixelů vytvoří vektor velikosti n^2.

3.3.2 Histogram

Tato poměrně nenáročná metoda dosahuje dobrých výsledků a zároveň generuje nízký počet složek výstupního vektoru. Některé příbuzné znaky však nedokáže odlišit a proto je dobré ji skombinovat ještě s další metodou. Princip spočívá v součtu všech černých pixelů v každém řádku a sloupci a je znázorněn na obrázku 3.4. Při velikosti $n \times n$ pixelů je vytvořen vektor příznaků s $2n$ prvky.

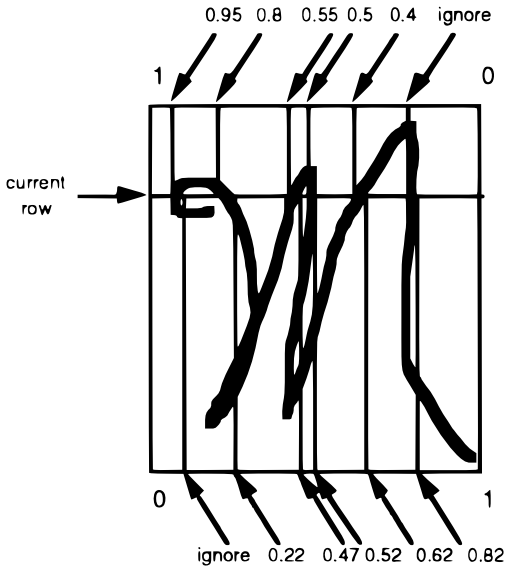


Obrázek 3.4: Použití histogramové metody na písmeno C

3.3.3 Polohy přechodů

Myšlenkou je vypočítat polohy přechodů z bílé barvy na černou v horizontálních a vertikálních směrech. Obrázek se prochází zprava doleva, zleva doprava, shora dolů a zdola nahoru [6].

V první fázi se počítají přechody v každém ze směrů. Každé získané číslo reprezentuje podíl vzdálenosti od počáteční hrany vzhledem k rozměru celého obrázku. Se vzdáleností hodnota klesá. Například při průchodu zleva doprava by měl přechod umístěný vlevo hodnotu blízkou 1.0, zatímco dál od levé hrany by se hodnota blížila nule. Vysvětleno je to na obrázku 3.5.



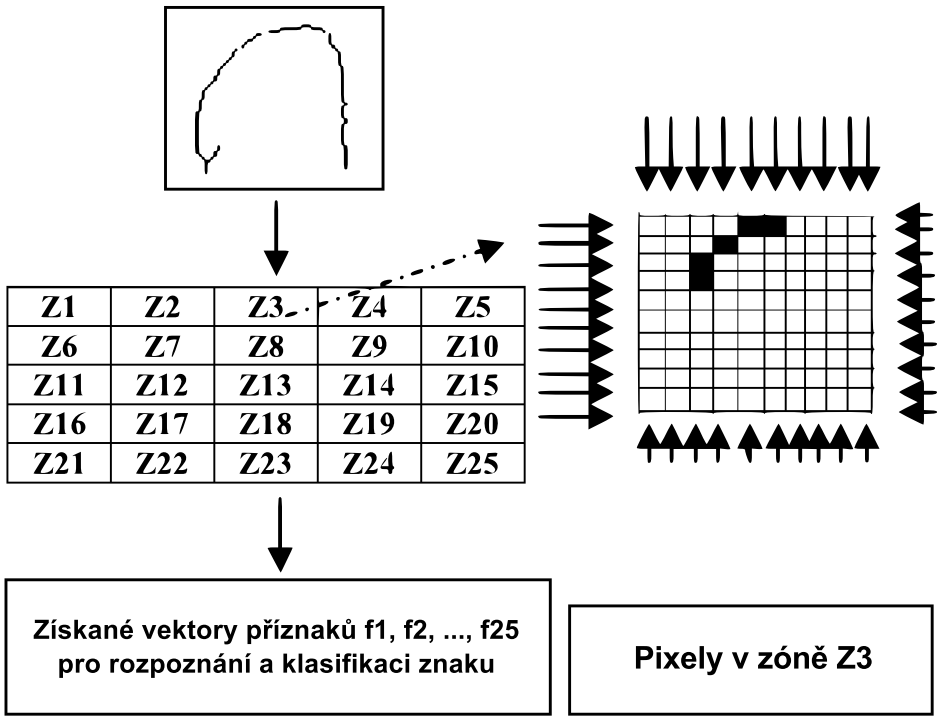
Obrázek 3.5: Výpočet relativních poloh přechodů na řádku [6]

Protože vektor příznaků musí mít konstantní velikost, počítá se pouze prvních M hodnot. V tomto případě je M rovno 3. Pokud je počet přechodů na řádku nebo sloupci menší než M, doplní se vektor nulami. Při velikosti obrázku $n \times n$ je délka vektoru příznaků $M \cdot n^2$, což je poměrně hodně. Proto se výsledná data ještě interpolují.

3.3.4 Dělení do zón

Tato metoda vyžaduje *skeletonizované znaky*. Mějme obrázek 28×28 pixelů, který rozdělíme do matice 4×4 zón. Každou zónu o rozměru 7×7 pixelů sekvenčně projdeme opět ve všech čtyřech směrech (obrázek 3.6). V každém řádku a sloupci spočítáme poměr vzdálenosti prvního černého pixelu od okraje zóny k celkovému rozměru obrázku. Mohou se vyskytnou řádky či sloupce, které neobsahují žádné černé pixely. V tomto případě je hodnota příznaku rovna nule [17].

Pokud je obrázek o velikosti $n \times n$ rozdělen do N zón s rozměry $z \times z$, získáme z něho vektor příznaků velikosti $4Nz$.



Obrázek 3.6: Extrakce příznaků z 25 zón o rozměrech 10×10 pixelů [17]

3.4 Trénování neuronové sítě

V této sekci bude vysvětleno, jak konkrétně probíhá učení nové neuronové sítě, s jakými daty se pracuje a jaké problémy při tomto procesu mohou nastat.

3.4.1 Trénovací data

Předtím, než se může začít síť učit, je třeba pro ni připravit vstupní sadu trénovacích dat. Z vybraných datasetů program postupně načítá obrázky ručně psaných znaků spolu s jejich popisky.

Na obrázky se použijí zvolené metody extrakce příznaků. Každá metoda načte binární data z obrázku, pomocí svého algoritmu z nich extrahuje potřebná data a ty vrátí ve formě vektoru s příznaky. Zároveň zajišťuje, aby hodnoty v tomto vektoru ležely v intervalu $\langle 0, 1 \rangle$. Pokud použijeme více těchto metod najednou, je třeba jejich výstup zřetězit do jediného vektoru, aby bylo možné příznaky přivést na vstup neuronové sítě.

Popisek obsahuje příslušné písmeno nebo číslici podle toho, o jaký znak na obrázku se jedná. Protože neuronová síť pracuje jak na vstupní tak na výstupní vrstvě pouze s vektory, i tento popisek musí být do této formy převeden. Číslicím **0**–**9** zůstává jejich vlastní hodnota. Písmena **A**–**Z** se namapují na číselnou množinu **11**–**35**. Když máme převedeny popisky na číselné hodnoty, vytvoříme pole o příslušné délce obsahující hodnoty -1. Pouze pozice odpovídající popisku bude obsahovat číslo jedna. Např. vektor pro písmeno B bude obsahovat jedničku na 11. pozici. Tímto dáme neuronové síti najevo, do které třídy znak patří. Velikost výstupního vektoru závisí na tom, zda síť bude rozpoznávat pouze číslice (10 prvků) nebo i písmena (36 prvků).
Všechna data, se kterými neuronová síť pracuje, se dělí do těchto kategorií:

- **Trénovací data**
  Tato sada se používá při trénování nové sítě. Jedná se zpravidla o největší část množiny všech vzorků. V tomto případě pro ni bylo vyčleněno kolem 70% vzorků.

- **Cross-validační data**
  Tyto data se používají taktéž v průběhu trénování. Po každé epoše (iteraci) se pomocí této sady vypočítá *cross-validační chyba*, která slouží k odhadnutí skutečné chyby aktuální konfigurace sítě. V této aplikaci není umístěna ve vlastním datasetu, ale odděluje se z trénovacích dat před samotným procesem učení. Tvoří asi 10% dostupných vzorků.

- **Testovací data**
  Když je síť natrénovaná, používá se tato množina k výpočtu její celkové úspěšnosti. K dosažení objektivního výsledku je nutné zajistit, aby trénovací a testovací data byly vzájemně disjunktní. Testovací množina obsahuje asi 20% všech dat.

3.4.2 Učení

Samotné učení nové sítě pracuje už pouze ze souborem trénovacích dat. Z těch se postupně načítají vektory příznaků, které se přivádějí na vstupní vrstvu, a referenční vektory, podle kterých se počítá aktuální chyba sítě a následně upravují váhy jednotlivých neuronů. Trénování má samo o sobě nekonečný průběh, proto je nutné definovat omezující kritéria, při kterých se algoritmus zastaví.

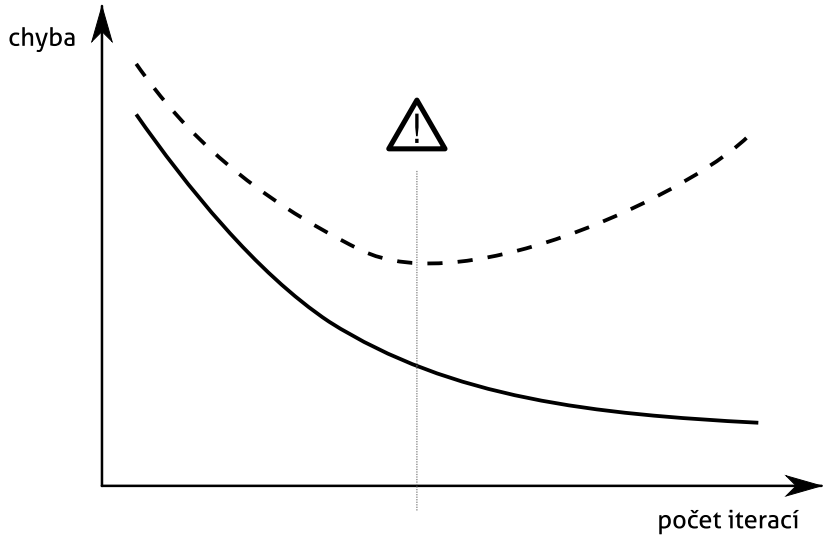
Jedním z nich je *průměrná kvadratická chyba*, v originále MSE (Mean Squared Error). Tato statistická chyba odhaduje úspěšnost predikce na trénovacích datech. V průběhu učení sítě tato chyba neustále klesá, dokud se nedostane pod námi stanovenou hodnotu.

Další je *počet epoch*, který určuje, po kolika krocích se trénování ukončí. Volba hodnoty tohoto údaje je ovlivněna jak rozsahem sítě, tak množstvím trénovacích dat. Čím lépe síť aproximuje hledanou funkci neboli čím rychleji chyba sítě konverguje, tím menší počet epoch je zapotřebí. Pro několik stovek vzorků není potřeba tolik iterací jako pro databázi MNIST s 60 000 vzorky.

Ochrana proti přetrénování

Nicméně nastavení požadované MSE ani maximálního počtu epoch nedokáže zajistit, aby se síť natrénovala optimálně. Pokud cyklus zastavíme předčasně, jednotlivé váhy neuronů nebudou správně oddělovat klasifikační množiny. Naopak pokud necháme proces běžet příliš dlouho, dojde k jevu zvanému *přetrénování* (overfitting). Síť v tomto stavu ztrácí schopnost **generalizace**. Tedy nedokáže rozpoznat jiná data než ta, na která byla natrénována.

K řešení tohoto problému je nejprve nutné pochopit, kdy k přetrénování dochází. V průběhu učení neuronové sítě po počátečním ustálení vah klesá její trénovací i cross-validační chyba. Postupem času, když už se síť nové informace naučila, pokles cross-validační chyby začne zpomalovat, až nakonec začne tato chyba růst. To je znázorněno na obrázku 3.7, kde je čárkovanou čarou zobrazena cross-validační chyba, plnou čarou trénovací chyba a trojúhelník s vykřičníkem označuje místo, kde je vhodné trénování zastavit.



Obrázek 3.7: Průběh trénovací a cross-validační chyby s místem, kde dochází k přetrénování

Pro detekci tohoto místa byl použit upravený algoritmus **New Bob** [19]. Nejprve je třeba zjistit, zda je síť již dostatečně *stabilní*. Hlavně v prvních epochách se velmi rychle mění trénovací i cross-validační chyba. Po nějaké době se váhy na jednotlivých neuronech ustálí a obě tyto chyby začnou klesat. Pokud chyby posledních 5 epoch měli klesající tendenci, algoritmus označí síť za stabilní.

Teprve poté se uplatní kritérium algoritmu New Bob. Pokud je rozdíl posledních dvou hodnot cross-validační chyby menší než 5%, *rychlost učení* zmenšíme na polovinu. Pokud se to stane podruhé, zastavíme trénování. Tím je zaručeno, že cross-validační chyba nezačne

růst a učení se ukončí v místě, kde křivka této chyby začne kopírovat horizontální směr.

Tento algoritmus je možné použít v případě dávkového trénování, kdy se váhy neuronů mění po celé epoše a chyba klesá pozvolna. Nicméně v průběhu experimentování bylo zjištěno, že lepší výsledky podává inkrementální trénování, které mění váhy po každém vzorku. Chyba tak velmi kolísá a algoritmus na ni nelze použít. Proto je zatím ochrana proti přetrénování deaktivována.

3.5 Segmentace textu

V optickém rozpoznávání textu hraje důležitou roli i schopnost rozdělit blok textu na menší jednotky (řádky, slova, znaky). Program byl navržen jako interaktivní aplikace, kde je možné na bílou plochu na jakékoliv místo napsat myší text v libovolném pořadí. Zde budou vysvětleny postupy, které byly použity k separaci textu z této kreslící oblasti.

3.5.1 Oddělení znaků

V této práci byl použit opačný postup než je obvyklé. Nejprve je celý obrázek získaný z kreslící oblasti rozdělen na jednotlivé znaky, které jsou až poté zařazeny do řádků. Pro rozpoznání celistvých segmentů je použito **semínkové vyplňování**.

To je možné aplikovat z toho důvodu, že jednotlivé tvary znaků jsou propojené a diakritiku neuvažujeme. V některých případech však může selhat. Například pokud by uživatel napsal písmeno skládající se z více tahů a nějakou jeho část by dostatečně nenapojil, bylo by rozděleno na více segmentů. Pro účely této aplikace je však tato metoda dostatečná.

Algoritmus prochází sekvenčně všechny pixely v obrázku. Pokud narazí na nějaký tmavý pixel, zavolá metodu, která záplavově vyplní všechny tmavé pixely v okolí, dokud nevyplní celou oblast. Přesněji tyto pixely zapíše do připravené *bitové masky* a původní černé pixely v obrázku smaže proto, aby nedošlo k duplikování stejných oblastí. Když se projde celým obrázkem, všechny získané masky se ořežou na velikost obsahu. Nakonec se zahodí ty segmenty, které mají plochu menší než práh K (v tomto případě $K = 100$). Jedná se nejčastěji o šum nebo zbytky smazaných znaků.

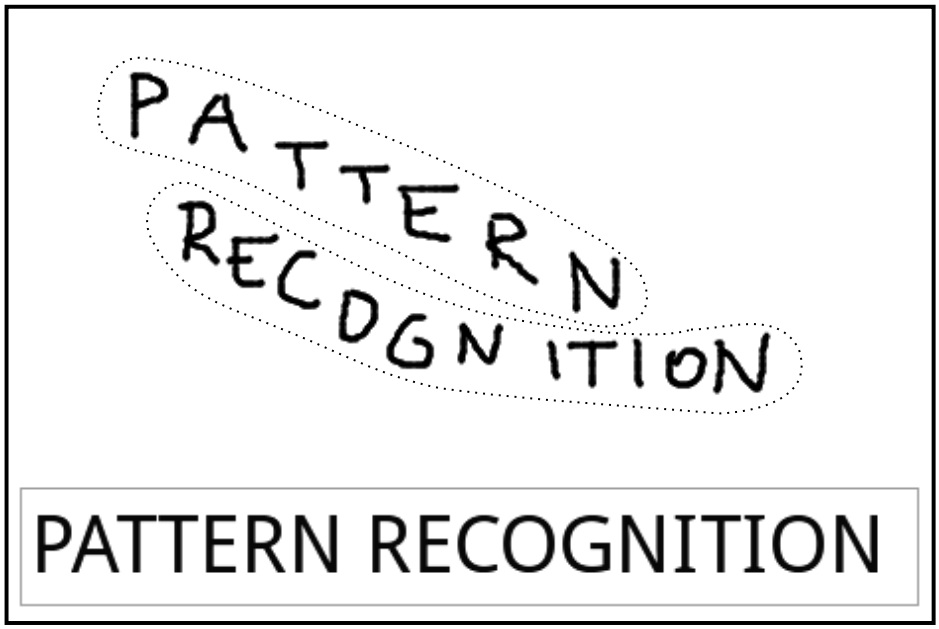
3.5.2 Řádkování

Získané segmenty znaků jsou seřazené podle pozice směrem zleva doprava a zbývá je zatřídit do řádků. Pro každý segment se projde seznam řádků, který je na počátku prázdný, a zjišťuje se, zda segment do řádku patří. Tedy zda platí tato tvrzení:

1. souřadnice x segmentu leží **za posledním znakem** kontrolovaného řádku
2. vertikální rozsah posledního znaku na řádku **se překrývá** se segmentem

Pokud jsou obě podmínky splněny, segment se vloží na konec řádku. V opačném případě je vytvořen nový řádek, do kterého je segment vložen. Na obrázku 3.8 je vidět, jak tento algoritmus označil jednotlivé řádky textu. I když se jedná o částečně deformované a skloněné řádky, byly rozlišeny a všechny znaky správně zařazeny.

Tento postup je alternativou k segmentační metodě oddělující jednotlivé řádky pomocí histogramu. Ta pracuje tak, že natáčí text v různých úhlech a zkoumá, zda jsou některé horizontální linie prázdné. Na tištěný text by tuto metodu šlo bez problémů použít. Na tomto příkladu by však nepracovala správně, protože zde je spodní řádek drobně zahnutý a zasahuje do vrchního.



Obrázek 3.8: Segmentace a rozpoznání dvou skloněných řádků

3.5.3 Mezery mezi slovy

Poslední úkol, který zbývá vyřešit, je přidat mezery na vhodná místa mezi znaky a tím řádek rozdělit na slova. Princip je založen na tom, že se předem spočítá **průměr šířky všech segmentů** na řádku. Poté se znovu projdou všechny segmenty. Mezera se vloží mezi ty, jejichž vzdálenost je větší než vypočtená průměrná šířka. Je tedy zapotřebí dvou průchodů polem segmentů. Metoda dává spolehlivé výsledky a vkládá mezeru skutečně tam, kde ji uživatel očekává.

Kapitola 4

Implementace

V této kapitole budou popsány konkrétní implementační detaily, použité jazyky, knihovny a způsob jejich vzájemného propojení.

4.1 Použité technologie

4.1.1 FANN

Pro manipulaci s neuronovými sítěmi byla vybrána knihovna FANN (Fast Artificial Neural Network Library). Byla zvolena především proto, že se jedná v současné době o nejvyspělejší volně dostupný open-source nástroj s aktivním vývojem. Knihovna je rychlá a napsaná v jazyku C s velkým výběrem trénovacích algoritmů (RPROP, Quickprop, Batch, Incremental) [11]. Obsahuje framework pro manipulaci s celými sítěmi a trénovacími daty. Mezi jinými poskytuje i rozhraní do jazyka Python. Na webových stránkách projektu [12] je možné nalézt obsáhlou dokumentaci, popisující veškeré moduly a funkce knihovny.

4.1.2 Qt4

Grafické uživatelské rozhraní aplikace bylo vytvořeno v toolkitu Qt4. I tato knihovna poskytuje rozhraní pod názvem PyQt4. Úroveň integrace knihovny do jazyka Python je na vysoké úrovni. Dochází k automatickému přetypování mezi řetězcovými typy `QString` a `str`, je možné vytvářet vlastní signály a obecná práce s frameworkem Qt je tak jednodušší. Při vývoji aplikace byla hojně využívána rozsáhlá referenční dokumentace [13].

4.1.3 Python

Před vlastní implementací projektu se bylo nutné rozhodnout pro vhodný programovací jazyk. Zvažovány byly především tyto dvě možnosti: C++ a Python. Oba jazyky jsou objektové, podporují tedy přirozené zapouzdřování kódu do tříd, a oba jsou schopny pracovat s knihovnami FANN a Qt4.

C++ má hlavní výhodu především ve své rychlosti, ale na druhou stranu je nutné se starat o správnou alokaci a uvolňování paměti a syntaxe jazyka je složitější.

Python má syntaxi jednoduchou, avšak dovoluje použít velmi efektivní programové konstrukce jako například *generátory*. Jedná se o jazyk interpretovaný a podává tedy nižší výkon než kompilovaný kód. Nicméně většina jeho knihoven je napsána v C/C++ a výkonově kritický kód je tak proveden dostatečně rychle. Spolu s možností prototypovat a ihned spouštět aplikace převážily výhody jazyka Python. Ten je zde použit jako mezivrstva

spojující dohromady všechny knihovny od uživatelského rozhraní až po systémové knihovny pro práci s vlákny.

Schopnosti interpretovaného jazyka byly využity i přímo v programu, kde je možné psát vlastní metody extrakce obrazových příznaků. Kód funkce je uložen jako prostý text. V případě potřeby je zkompilován do bytekódu pomocí funkce `exec()` a proveden.

4.1.4 Python Imaging Library

Tento modul [16] do jazyka Python přidává schopnosti pro práci s rastrovými obrázky. Obsahuje třídy pro reprezentaci barev, filtrů, vykreslení písem a podporuje velké množství formátů souborů. Tato knihovna je využita především pro zpracování formulářů a také pro manipulaci s obrazovými informacemi v metodách extrakce příznaků.

Obrázky se v aplikaci zobrazují jako instance třídy `QPixmap`. Pro práci s nimi je ale potřebujeme převést na instance třídy `Image` z modulu PIL. Ve zdrojovém kódu algoritmu 4 je ukázka tohoto převodu, který se používá pro získání obrázku z interaktivní kreslící plochy. Nejprve se převede `QPixmap` na `QImage` s 24-bitovým barevným RGB prostorem. Poté se z obrázku získají metodou `bits()` surová obrazová data, která se načtou metodou `frombuffer()` do nové instance třídy `Image`. Nakonec se ještě `Image` převede do 8-bitových odstínů šedi a vrátí se.

Algoritmus 4 Převod z QPixmap na PIL Image

```
def getImage(pixmap):
   qimage = pixmap.toImage().convertToFormat(QtGui.QImage.Format_RGB888)
   buffer = qimage.bits().asstring(qimage.numBytes())
   size = qimage.width(), qimage.height()
   im = Image.frombuffer("RGB", size, buffer, "raw", "RGB", 0, 1)
   im = im.convert("L")
   return im
```

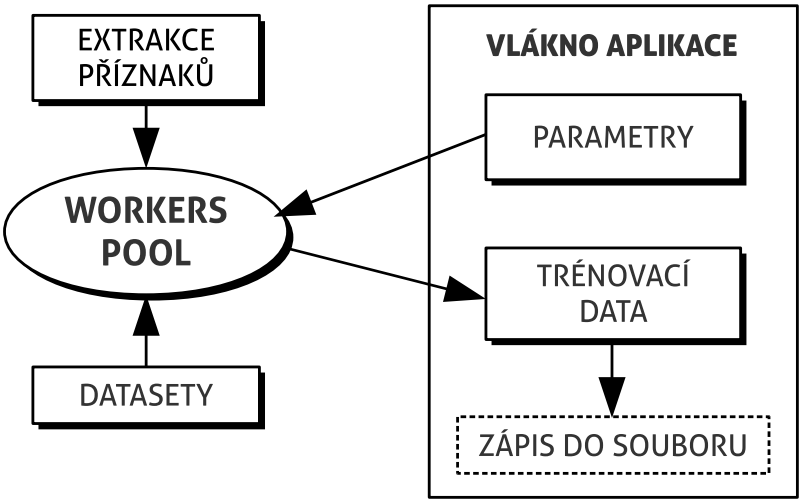
4.2 Vícevláknové provádění

Generování příznaků ze vzorků je poměrně náročné na výkon počítače. Obzvlášť, když jsou metody extrakce příznaků nejprve kompilovány do bytecodu a poté interpretovány virtuálním strojem Pythonu. Proto jsem se rozhodl využít výhod vícejádrových procesorů a vytváření trénovacích dat napsat vícevláknově. Samotné trénování neuronové sítě je implementováno knihovnou FANN a nemůže být nijak urychleno.

Knihovnu pro práci se systémovými vlákny Python poskytuje, nicméně zde existuje jedno omezení. Kvůli globálnímu zámku interpretu *GIL* (Global Interpreter Lock) je možné provádět v daném čase vždy pouze jedno vlákno [1]. Proto bylo použito systémových **procesů**, což jsou nezávislé instance s odděleným paměťovým prostorem.

Ze všeho nejdříve je vytvořen tzv. **pool of workers** [15]. Jedná se o seznam vytvořených procesů, které čekají na parametry, aby mohly začít vykonávat nějakou funkci. Když proces zadanou práci dokončí, neukončí se, ale opět čeká na signál s dalšími instrukcemi. Předpoklad je, že optimální velikost poolu je rovna počtu jader procesoru, který se dá zjistit metodou `multiprocessing.cpu_count()`.

Poté jsou všechny datasety určené ke zpracování rozděleny na dávky po 200 vzorcích a tyto požadavky jsou uloženy do pole parametrů. Funkcí `Pool.imap_unordered()` předáme parametry struktuře `Pool`, která se postará o postupné volání worker procesů a vracení získaných dat zpět hlavnímu vláknu aplikace. Každý worker zpracuje zadanou dávku vzorků a vrátí serializovaná trénovací data, která program zapíše do souboru. Tento postup je schematicky znázorněn na obrázku 4.1.



Obrázek 4.1: Implementace vícevláknové extrakce příznaků

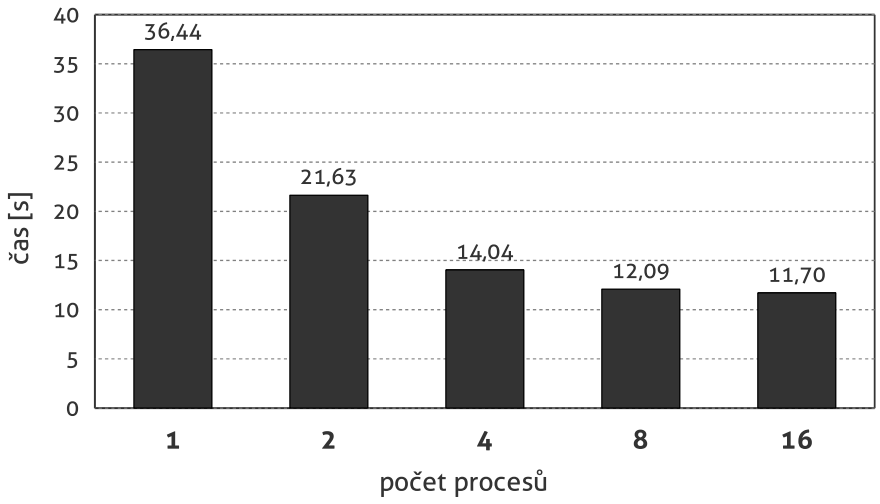
Kapitola 5

Experimentování

5.1 Vytváření trénovacích dat

Protože je tato operace implementována paralelně, bylo by dobré prozkoumat, jak se chová na vícejádrových procesorech a jakého reálného zrychlení bylo dosaženo. V testu byla použita metoda extrakce příznaků *Polohy přechodů*, která byla aplikována na sadu 10000 vzorků databáze MNIST.

Jako běhové prostředí posloužil čtyřjádrový procesor **Intel Core i7 740QM** s frekvencí 1,73 GHz, 6 MB L3 Cache a podporou až 8 běžících vláken. Bylo provedeno celkem pět testů vždy s jedním, dvěma, čtyřmi, osmi a šestnácti souběžně prováděnými procesy. Časy běhu testů jsou uvedeny v grafu 5.1.



Obrázek 5.1: Doba vytváření trénovacích dat při různém počtu procesů

Původní předpoklad byl, že nejkratšího času se dosáhne, pokud bude počet procesů roven počtu jader procesoru. Tento předpoklad se potvrdil. I když je nejrychlejší výsledek s 16 procesy, je téměř totožný jako čas běhu 8 procesů na osmi logických jádrech.

K velké režii dochází při posílání dat mezi procesem a hlavním vláknem programu. Tyto data se původně posílala ve formě pole, které se však muselo před odesláním serializovat a poté znovu vytvořit. Velkého urychlení bylo dosaženo tím, že se tyto data vracejí ve formě

řetězce, který je ihned zapisován do souboru. Řetězec je reprezentován určitým rozsahem bajtů v paměti a tak je jeho přenos mezi procesy mnohem jednodušší.

5.2 Úspěšnost rozpoznávání

Jedním z hlavních cílů této práce bylo otestovat úspěšnost klasifikátoru na reálných datech. K pokusu byly vybrány dvě různé sady dat. První sadu tvoří 32 833 trénovacích a 8 245 testovacích vzorků, které byly získány z formulářů vyplněných studenty FIT VUT. Druhá sada, databáze MNIST, obsahuje 60 000 trénovacích a 10 000 testovacích dat, která pochází od zaměstnanců firmy Census Bureau a vysokoškolských studentů [10].

Obě sady byly upraveny tak, aby obsahovaly obrázky v šedé škále o velikosti 28×28 pixelů. Před použitím byly obrázky prahovány s parametrem $T = 200$, jejich obsah zmenšen na 24×24 pixelů. Znak se však neroztahuje celý, ale je zachován jeho poměr stran, protože nese důležitou informaci pro extrakci příznaků. Pokud by byly všechny znaky stejně široké, úspěšnost rozpoznávání by byla mnohem menší. Znak je také možné zarovnat na střed, a to těmito způsoby:

- **Bounding box**
  Naleznou se hranice obsahu zleva, zprava, shora a zdola a získaný obdélník se zarovná na střed.

- **Center of mass**
  Vypočítá se těžiště pixelů a tento bod se posune na střed obrázku. Těžiště je takové místo, od něhož doleva leží stejný počet černých pixelů jako doprava a nad ním leží stejný počet černých pixelů jako pod ním.

Pomocí Center of mass byl původně zarovnán MNIST a metodou Bounding box data získaná z formulářů. V průběhu testování se ukázalo, že zarovnání pomocí hranice obsahu **významně snižuje chybovost** při rozpoznávání pomocí neuronové sítě. Proto byly i poté všechny znaky sady MNIST zarovnány pomocí této metody. Tato jednoduchá operace může zvýšit úspěšnost téměř o 20 %, jak je to demonstrováno v tabulce 5.1.

Z důvodu dlouhé doby trénování na všech vzorcích byla vytvořena zmenšená sada. Z databáze MNIST i z formulářů bylo odděleno 3000 trénovacích a 1500 testovacích vzorků, na kterých byl experiment proveden. Ve všech pokusech bylo trénování ukončeno po 50-ti epochách, což je dostatečný počet vzhledem k velikosti použité sady. Postup provádění pokusů je zaznamenán v tabulkách 5.1 a 5.2. Ve sloupci úspěch je uvedeno procento správně rozpoznaných znaků. Sloupce IN, HID a OUT znamenají počet neuronů ve vstupní, skryté a výstupní vrstvě.

úspěch	IN	HID	OUT	zarovnání	extrakce příznaků
52,0 %	56	36	10	center of mass	histogram
52,6 %	784	400	10	center of mass	pixely
59,4 %	196	100	10	center of mass	pixely 14×14
72,4 %	56	36	10	bounding box	histogram
84,4 %	448	256	10	bounding box	dělení do zón
88,0 %	84	47	10	bounding box	polohy přechodů
88,8 %	140	75	10	bounding box	polohy přechodů, histogram

Tabulka 5.1: Experimentování na zmenšené sadě MNIST

úspěch	IN	HID	OUT	zarovnání	extrakce příznaků
48,8 %	56	46	36	bounding box	histogram
51,9 %	784	410	36	bounding box	pixely
54,3 %	252	160	36	bounding box	histogram, pixely 14×14
53,9 %	252	256	36	bounding box	histogram, pixely 14×14
58,4 %	448	256	36	bounding box	dělení do zón
72,3 %	84	60	36	bounding box	polohy přechodů
73,4 %	84	64	36	bounding box	polohy přechodů
72,5 %	140	88	36	bounding box	polohy přechodů, histogram

Tabulka 5.2: Experimentování na zmenšené sadě vzorků z formulářů

Úspěšnost rozpoznávání vzorků MNIST byla vyšší především z toho důvodu, že tato sada obsahuje pouze číslice. Je tedy potřeba každou z nich zařadit pouze do jedné z deseti tříd. Naproti tomu formuláře obsahují jak číslice, tak všechna písmena, které celkem tvoří 36 tříd. Je zde tedy daleko větší pravděpodobnost, že se neuronová síť „splete“.

Poměrně intenzivně bylo také experimentováno s **velikostí skryté vrstvy**. V některých případech vyšší počet neuronů v této vrstvě zvýší úspěšnost rozpoznání. Když jich je ale příliš mnoho, úspěšnost klesá. Nebyl nalezen žádný vztah, kterým by se dal ideální počet neuronů skryté vrstvy vypočítat. Experimentálně bylo zjištěno, že poměrně dobré výsledky dává *aritmetický průměr* velikosti vstupní a výstupní vrstvy. Tento vzorec je použit i v programu pro automatický výpočet, nicméně uživatel má možnost parametr změnit.

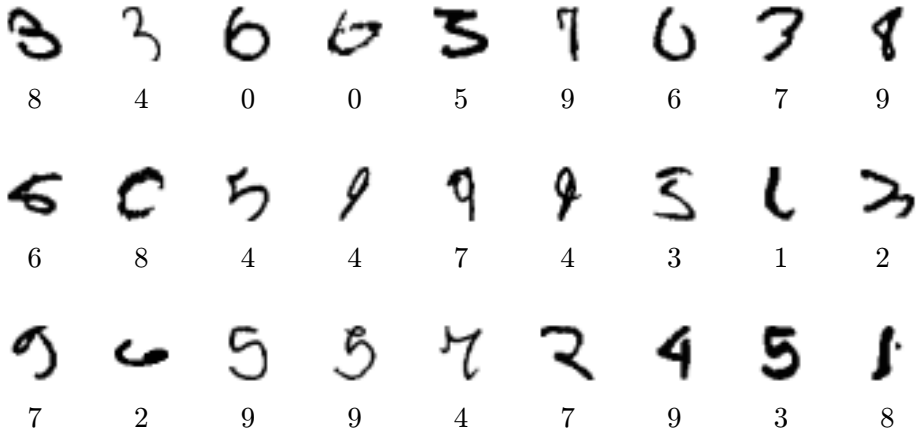
Postupně byly testovány všechny implementované extrakce příznaků. Ukázalo se, že metoda *Histogram* je velmi rychlá a poměrně přesná. Téměř stejnou úspěšnost má i metoda *Pixely*, která však generuje nepřípustně dlouhý vektor příznaků a tak významně zpomaluje trénování. Určitým řešením je zmenšit před použitím této metody obrázek na polovinu (14×14 pixelů), čímž se zkrátí vektor příznaků a dokonce i vzroste úspěšnost rozpoznání. Metoda *Dělení do zón* je podle výsledků opět o něco přesnější, ale je náročná na výpočet a také vytváří velký vektor příznaků. Jako nejlepší se jeví metoda *Polohy přechodů*, která produkuje krátký vektor, není tolik výpočetně náročná jako metoda předchozí a dosahuje vysoké úspěšnosti. Ona samotná nebo v kombinaci s metodou *Histogram* správně rozpoznají největší počet vzorků.

Když byla nalezena nejúspěšnější kombinace metod na zmenšené sadě vzorků, byla použita i při trénování sítě na celých sadách. Vzhledem k většímu počtu vzorků byl zvolen i

úspěch	sada	IN	HID	OUT	epoch	extrakce příznaků
96,7 %	MNIST	140	96	10	250	polohy přechodů, histogram
85,2 %	Formuláře	140	128	36	200	polohy přechodů, histogram

Tabulka 5.3: Experimentování s celými sadami

větší počet epoch: 250 pro MNIST a 200 pro sadu z formulářů (viz. tabulka 5.3). Tím se prodloužila i doba trénování z několika vteřin na několik desítek minut. Úspěšnost rozpoznání se pohybuje okolo 90 %.



Tabulka 5.4: Některé chybně rozpoznané vzorky sady MNIST; každý obrázek pod sebou obsahuje rozpoznanou číslici

V tabulce 5.4 jsou uvedeny některé případy ze sady MNIST, na kterých klasifikátor selhal. U každého vzorku je na spodní straně zobrazena rozpoznaná číslice. Je možné si všimnout, že některé obrázky jsou neúplné nebo zdeformované tak, že ani člověk možná na první pohled neodhadne jejich obsah. Právě kvůli výskytu takovýchto případů je nereálné dosáhnout 100% úspěšnosti.

Kapitola 6

Závěr

V rámci této práce byl prozkoumán princip funkce umělých neuronových sítí a jejich aplikace na rozpoznávání vzorů. Díky knihovně FANN bylo možné s těmito sítěmi snadno pracovat a detailně nastavovat jednotlivé parametry k dosažení požadovaných vlastností. Pro extrakci příznaků ze znaků bylo implementováno několik metod, z nichž nejlepší výsledky dává algoritmus **Polohy přechodů**. Ukázalo se, že i skriptovací jazyk Python je vhodný na tvorbu rozsáhlejšího projektu, pokud je výkonově kritická část obsažena v kompilovaných knihovnách.

Vznikla aplikace s grafickým uživatelským rozhraním, která umožňuje parsovat vyplněné formuláře a vytvářet z nich datové sady, na jejich základě trénovat nové neuronové sítě a testovat existující sítě. Ve vestavěném editoru lze psát vlastní metody pro získání obrazových příznaků, zkoušet interaktivní rozpoznávání myší psaného písma a také spravovat všechny vytvořené soubory. Program využívá multiplatformní knihovny a měl by být bez problémů spustitelný na operačních systémech Windows i Linux.

Byly také provedeny poměrně úspěšné experimenty s přesností klasifikace na reálných databázích. Na standardní sadě MNIST byla dosažena úspěšnost **96,7 %**, což je výsledek srovnatelný s nejlepší úspěšností publikovanou na webu MNIST [10] při použití třívrstvé neuronové sítě. Tato hodnota je 98,47 %. Na sadě získané z formulářů, které obsahují číslice i písmena, bylo dosaženo úspěšnosti **85,2 %**.

V současné době aplikace rozpoznává znaky psané hůlkovým písmem. V případě dalšího vývoje by bylo vhodné do této množiny zahrnout nejprve písmo psané tzv. kurzívou a posléze i písmo psací. Bylo by nutné navrhnout novou metodu segmentace znaků, která by dokázala oddělit spojená písmena od sebe a zároveň zachovat jejich diakritiku. Pro profesionálnější použití, kde by bylo za úkol rozpoznávání celých slov nebo vět, by také bylo vhodné použití *jazykového modelu*. Ten by na základě statistik výskytu slov obsažených v určitém trénovacím korpusu byl schopen opravit chyby vzniklé špatným rozpoznáním některých znaků v rámci slova nebo slovního spojení a výrazně zlepšit úspěšnost OCR na úrovni těchto vyšších celků.

Literatura

[1] jj Behrens, S.: Concurrency and Python. *Dr. Dobb's Journal*, únor 2008: str. 2.
URL http://drdobbs.com/open-source/206103078?pgno=2

[2] Bishop, C. M.: *Pattern recognition and machine learning*. Springer-Verlag New York, 2006, ISBN 978-0387-31073-2, 740 s.

[3] Cychosz, J. M.: *Efficient binary image thinning using neighborhood maps*. San Diego, CA, USA: Academic Press Professional, Inc., 1994, ISBN 0-12-336155-9, s. 465–473.

[4] Devijver, P.; Kittler, J.: *Pattern Recognition: A Statistical Approach*. Prentice-Hall, 1982.

[5] Øivind Due Trier; Taxt, T.; Jain, A. K.: Feature extraction methods for character recognition - A survey. *Pattern recognition*, ročník No.29, č. 4, 1996: s. 641–662, ISSN 0031-3203/96.

[6] Gader, P.; Mohamed, M.; Chiang, J.-H.: Handwritten word recognition with character and inter-character neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics*, ročník 27, únor 1997: s. 158–164, ISSN 1083-4419.

[7] Hornik, K.; Stinchcombe, M.; White, H.: Multilayer feedforward networks are universal approximators. *Neural Netw.*, ročník 2, červenec 1989: s. 359–366, ISSN 0893-6080.

[8] Hricko, J.: *Rozpoznávání ručně psaných číslic pomocí support vector machines*. Bakalářská práce, Vysoké učení technické v Brně, 2010.

[9] Jan, J.: *Číslicová filtrace, analýza a restaurace signálů*. VUTIUM, 2002, ISBN 80-214-1558-4, 427 s.

[10] LeCun, Y.; Cortes, C.: THE MNIST DATABASE of handwritten digits. [cit. 2011-04-30].
URL http://yann.lecun.com/exdb/mnist/

[11] Nissen, S.: Neural Networks Made Simple. *Software Developer's Journal*, únor 2005: s. 14–19.
URL http://fann.sf.net/fann_en.pdf

[12] Nissen, S.: Fast Artificial Neural Network Library. [cit. 2011-05-03].
URL http://leenissen.dk/fann/wp/

[13] Nokia Corporation: Qt Reference Documentation. [cit. 2011-05-04].
URL http://doc.qt.nokia.com/latest/

[14] Orlinsky, K.: Rewiring the Brain. [cit. 2011-05-11].
URL
http://stochasticscientist.blogspot.com/2010/01/rewiring-brain.html

[15] Python Software Foundation: multiprocessing: Process-based threading interface. [cit. 2011-05-04].
URL http://docs.python.org/library/multiprocessing.html

[16] PythonWare: Python Imaging Library. [cit. 2011-05-04].
URL http://www.pythonware.com/products/pil/

[17] Rajashekararadhya, S.; Ranjan, P.: Zone based Feature Extraction Algorithm for Handwritten Numeral Recognition of Kannada Script. *Advance Computing Conference, 2009*, březen 2009: s. 525–528.

[18] Saidl, J.: *Vizualizace jako nástroj studia chování modelu přírodních systémů*. Diplomová práce, České vysoké učení technické v Praze, 2006.

[19] Schwarz, P.; Burget, L.; Černocký, J.; aj.: Klasifikace a rozpoznávání: Umělé neuronové sítě a support vector machines. [cit. 2011-05-03].
URL http://www.fit.vutbr.cz/study/courses/IKR/public/prednasky/

[20] WWW stránky: Project Gutenberg. [cit. 2011-04-26].
URL http://www.gutenberg.org/

Příloha A

Obsah CD

Optické médium přiložené k práci obsahuje tyto soubory a složky:

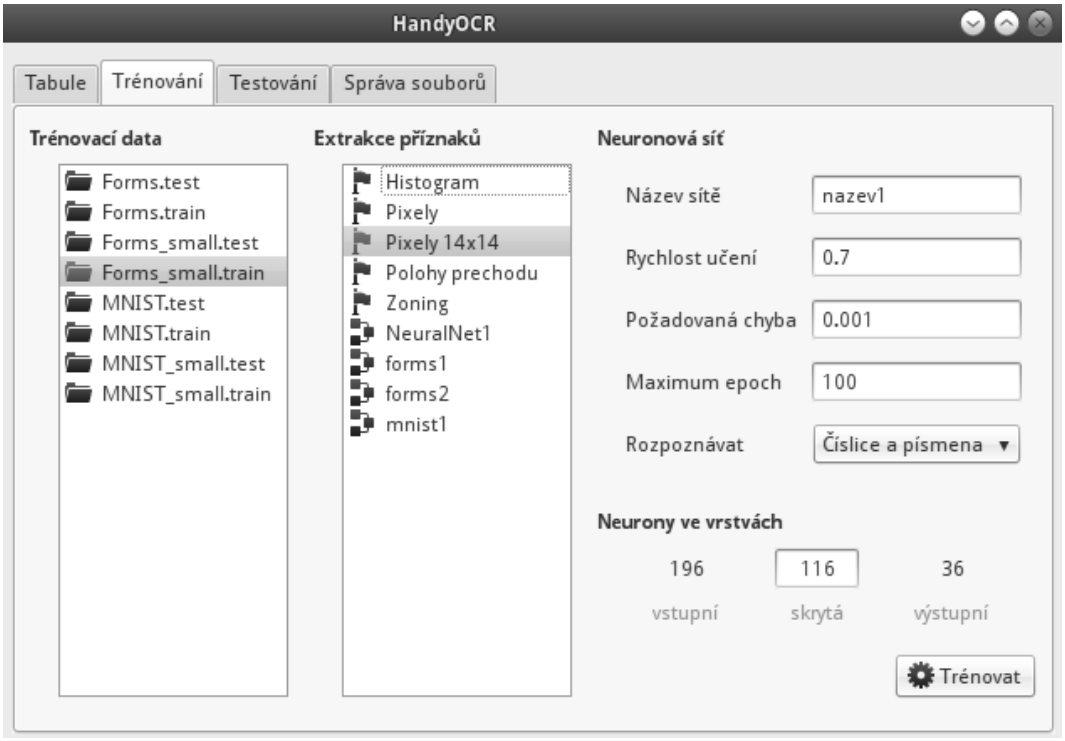
README	stručný návod k ovládání aplikace
INSTALL	minimální požadavky a postup instalace
report.pdf	tato technická zpráva
report/	zdrojové soubory a obrázky pro vytvoření této zprávy
app/	složka s vlastní aplikací
app/Datas/	trénovací a testovací datové sady
app/Features/	metody extrakce příznaků
app/Forms/	vyplněné naskenované formuláře
app/Nets/	neuronové sítě
fann/	zdrojové kódy knihovny FANN

Příloha B

Screenshoty aplikace

Obrázek B.1: Interaktivní oblast pro psaní a okamžité rozpoznávání

V levé části obrázku B.1 je vidět **ikona tužky** sloužící k výběru kreslícího nástroje a **ikona gumy** aktivující nástroj pro mazání. Vlevo dole je tlačítko s **ikonou grafu**, které zobrazí statistiky rozpoznání jednotlivých znaků. Převážnou část okna zabírá kreslící plocha. Pod ní je umístěno tlačítko **Smazat**, které vymaže celé plátno, box s výběrem použité neuronové sítě a tlačítko **Segmentace**, které barevně zvýrazní jednotlivé segmenty znaků. Pod těmito tlačítky se zobrazuje rozpoznaný text.



Obrázek B.2: Nastavení parametrů pro trénování nové sítě

Obrázek B.2 zobrazuje formulář pro trénování nové sítě. V levém sloupci je možné vybrat datové sady, které se pro trénování použijí. Ve sloupci uprostřed se nacházejí metody extrakce příznaků, kterými mohou být i existující neuronové sítě. V pravé části se zadává název sítě, její parametry a kritéria omezující trénování. Vpravo dole je poté možné vidět, kolik neuronů v jednotlivých vrstvách daná síť bude obsahovat. Počet skrytých neuronů je možné měnit. Tlačítko **Trénovat** celý proces spustí.